# Evolution of the Milieu Approach for Software Development for the Polymorphous Computing Architecture Program

Yoginder S. Dandass
Mississippi State University

Theodore Bapty
Vanderbilt University

Ben Abbott
Southwest Research Institute

Anthony Skjellum
MPI Software Technology, Inc.

Charles Summey
MPI Software Technology, Inc.

Hong Yuan
MPI Software Technology, Inc.

A key goal of the DARPA Polymorphous Computing Architectures (PCA) program is to develop reactive closed-loop systems that are capable of being dynamically reconfigured in order to respond to changing mission scenarios. This is in contrast to current systems that are fixed in nature and rely on architecture and software optimizations targeted for specific missions. In order to accomplish this objective, a number of “malleable” processing elements, runtime support software, compilers, and application development tools are being developed by a variety of research teams. Application software development for PCA systems is expected to be particularly challenging because of the need to respond to rapid changes in the mission requirements and environment. The authors of this talk have formed a team that is focusing its effort on developing application modeling tools and middleware libraries that assist in managing the complexity of developing, deploying, and maintaining PCA applications.

The four high-level research objectives of this effort are as follows:

- Study, prototype and develop a modeling language for streaming and threaded resources and components,
- Perform Design Space exploration to enable PCA scheduling (including use of AI techniques),
- Achieve System Synthesis and Generation (including performance monitoring and feedback) to support, and
- Accomplish Dynamic Reconfiguration study and support for PCA.

Design space exploration/navigation and optimization techniques are used to map the application and specifications to component and resource implementations. Generation includes configuration of runtime dynamic resource managers and local system optimization to ensure correct and efficient morphs in dynamic, non-pre-verified configurations. Online composition of these verified/optimized designs implements the new target morph configurations.

Because it is impossible to pre-generate all potential configurations of a PCA system, online configuration is necessary. Since the typical design space for an application is large, online navigation of the entire design space proves infeasible. Instead, the design space is pre-digest down to restricted subspaces. These subspaces offer the flexibility of optimization among architecture, resource, and application variations with the ability to find solutions rapidly, in a guaranteed time period.

During system generation, allocated and unused resources along with mission and application requirement constraints are collected in the form of *metadata* that are compiled into the optimized system schedule thereby providing an annotated interface for various (perhaps competing) applications requesting resource configuration changes during runtime.

Metadata compiled by the optimization and scheduling tools (as depicted in Figure 1) drives the morphing performed by a PCA system. This compiled metadata is assembled from metadata inputs from the resource modeling, application and component modeling, and programming-style modeling tools. The compiled metadata also partitions the optimization search space, thereby reducing the time required by the runtime manager to dynamically optimize resource allocations.

In many cases, the compiled metadata will explicitly enumerate a number of optimal resource allocation schedules generated offline in order to expedite mode changes. The software architecture under investigation also provides for run-time system performance monitoring by the resource manager and allows the runtime system to perform limited online optimization in response to dynamic application requests and feedback from the hardware-based performance monitoring services.

Modeling allows explicit representation of alternative components, allowing functionally equivalent components to be included in the design when models are constructed. Alternatives are resolved through design-space exploration.

# Report Documentation Page

*Form Approved*
*OMB No. 0704-0188*

| | | |
|---|---|---|
| 1. REPORT DATE<br>**20 AUG 2004** | 2. REPORT TYPE<br>**N/A** | 3. DATES COVERED<br>**-** |

| | |
|---|---|
| 4. TITLE AND SUBTITLE<br>**Evolution of the Milieu Approach for Software Development for the Polymorphous Computing Architecture Program** | 5a. CONTRACT NUMBER |
| | 5b. GRANT NUMBER |
| | 5c. PROGRAM ELEMENT NUMBER |
| 6. AUTHOR(S) | 5d. PROJECT NUMBER |
| | 5e. TASK NUMBER |
| | 5f. WORK UNIT NUMBER |
| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)<br>**Mississippi State University; MPI Software Technology, Inc.; Vanderbilt University** | 8. PERFORMING ORGANIZATION REPORT NUMBER |
| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) | 10. SPONSOR/MONITOR'S ACRONYM(S) |
| | 11. SPONSOR/MONITOR'S REPORT NUMBER(S) |

12. DISTRIBUTION/AVAILABILITY STATEMENT
**Approved for public release, distribution unlimited**

13. SUPPLEMENTARY NOTES
**See also ADM001694, HPEC-6-Vol 1 ESC-TR-2003-081; High Performance Embedded Computing (HPEC) Workshop (7th)., The original document contains color images.**

14. ABSTRACT

15. SUBJECT TERMS

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| a. REPORT<br>**unclassified** | b. ABSTRACT<br>**unclassified** | c. THIS PAGE<br>**unclassified** | **UU** | **39** | |

**Standard Form 298 (Rev. 8-98)**
Prescribed by ANSI Std Z39-18

Component models contain information about how a component may morph, including the types of morph requests a component will issue, bounds on those requests (min/max number of processors a component will request), types of morphs that can be requested of the component (morph constraints), and so on.

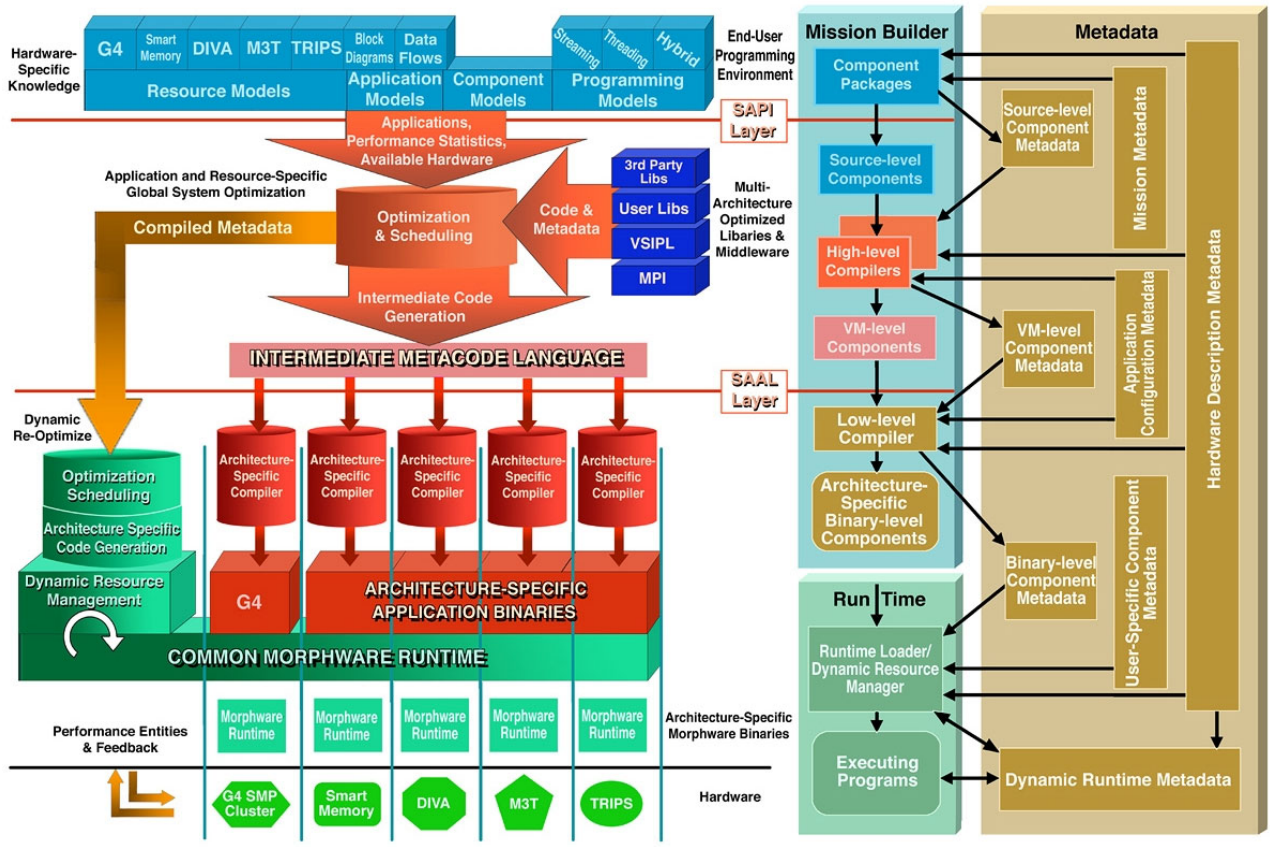

Hardware-Specific Knowledge
G4
Smart Memory
DIVA
M3T
TRIPS
Block Diagrams
Data Flows
Streaming
Threading
Hybrid
Resource Models
Application Models
Component Models
Programming Models
End-User Programming Environment
SAPI Layer
Applications, Performance Statistics, Available Hardware
Application and Resource-Specific Global System Optimization
Compiled Metadata
Optimization & Scheduling
Code & Metadata
3rd Party Libs
User Libs
VSIPL
MPI
Multi-Architecture Optimized Libaries & Middleware
Intermediate Code Generation
INTERMEDIATE METACODE LANGUAGE
SAAL Layer
Dynamic Re-Optimize
Optimization Scheduling
Architecture Specific Code Generation
Dynamic Resource Management
Architecture-Specific Compiler
G4
ARCHITECTURE-SPECIFIC APPLICATION BINARIES
COMMON MORPHWARE RUNTIME
Morphware Runtime
Performance Entities & Feedback
Architecture-Specific Morphware Binaries
G4 SMP Cluster
Smart Memory
DIVA
M3T
TRIPS
Hardware
Mission Builder
Component Packages
Source-level Components
High-level Compilers
VM-level Components
Low-level Compiler
Architecture-Specific Binary-level Components
Run Time
Runtime Loader/ Dynamic Resource Manager
Executing Programs
Metadata
Source-level Component Metadata
Mission Metadata
Hardware Description Metadata
VM-level Component Metadata
Application Configuration Metadata
Binary-level Component Metadata
User-Specific Component Metadata
Dynamic Runtime Metadata

Figure 1: Overview of the Milieu Application Construction Environment

Once design space navigation has been performed, a scheduling optimization pass performs a fine-grained analysis on the components mapped to each node to determine the proper schedule for each node. Combinatorial optimization techniques developed in the areas of operations research and artificial intelligence are under investigation in order to systematically and efficiently search the solution space for optimal schedules. Scheduling algorithms based on variants of branch-and-bound and other AI-based search techniques to construct schedules using heuristics are also being investigated. Simulated annealing and genetic/evolutionary programming techniques are utilized for evolving near-optimal schedules for those problems that cannot be solved using heuristics alone. In many dynamic real-time systems it is impossible to anticipate and plan for all possible combinations of workloads a-priori. In such systems, neural network and reinforcement learning technologies can be applied to construct scheduling algorithms that learn effective scheduling strategies during system runtime.

The compiled application constraints support and coordinate individual application performance perturbation in a centralized, priority-based, fashion. As such, violation of required guaranteed real-time deadlines are avoided through the constraint verification process. For example, if a non-real-time application requests sharing of resources that would cause a currently active real-time application on the same PCA hardware to compromise its timing constraints, the non-real-time application may be denied its resource request. This implies that the non-real-time application was given an importance lower than that of the real-time application. The actual importance of the various applications will be based on an application suggested and common morphware runtime constrained priority number. Thus, even if a task is non-real-time, but is very important, a system operator may guide it to become "most important" (even to the cost of missing deadlines in other applications).

In addition to maintaining hard resource constraints, the dynamic resource manager also maintains and adjusts soft constraints by collating performance metrics feed back from currently running applications. These current and

Abstract Presentation Back to Agenda Next Abstract

historic performance metrics along with the compiled constraints supplied directly through the model guide online system optimization during runtime. Thus, an application that did not directly request an exact architecture could be bounced through a variety of shapes as the overall PCA system attempts to “tune” itself.

This talk will provide an overview of the approach use by the authors of the talk to manage the problem of PCA application development through the use of application modeling tools and middleware libraries. The talk also introduces the critical elements of the required tool chain that will lead the successful deployment of reconfigurable software components.

Abstract Presentation

Back to Agenda

Next Abstract

# Milieu Approach for Software Development for the PCA Program

Yoginder Dandass
Mississippi State University

Anthony Skjellum
Charles Summey
Hong Yuan
MPI Software Technology, Inc.

Ted Bapty
Vanderbilt University
Ben Abbott
Southwest Research Institute

# Organization

- Goals
- Software Architecture
- Build Chain
- Metadata
- Dynamic Resource Management
- Software Components
- Prototype Results
- Future Work
- Conclusion

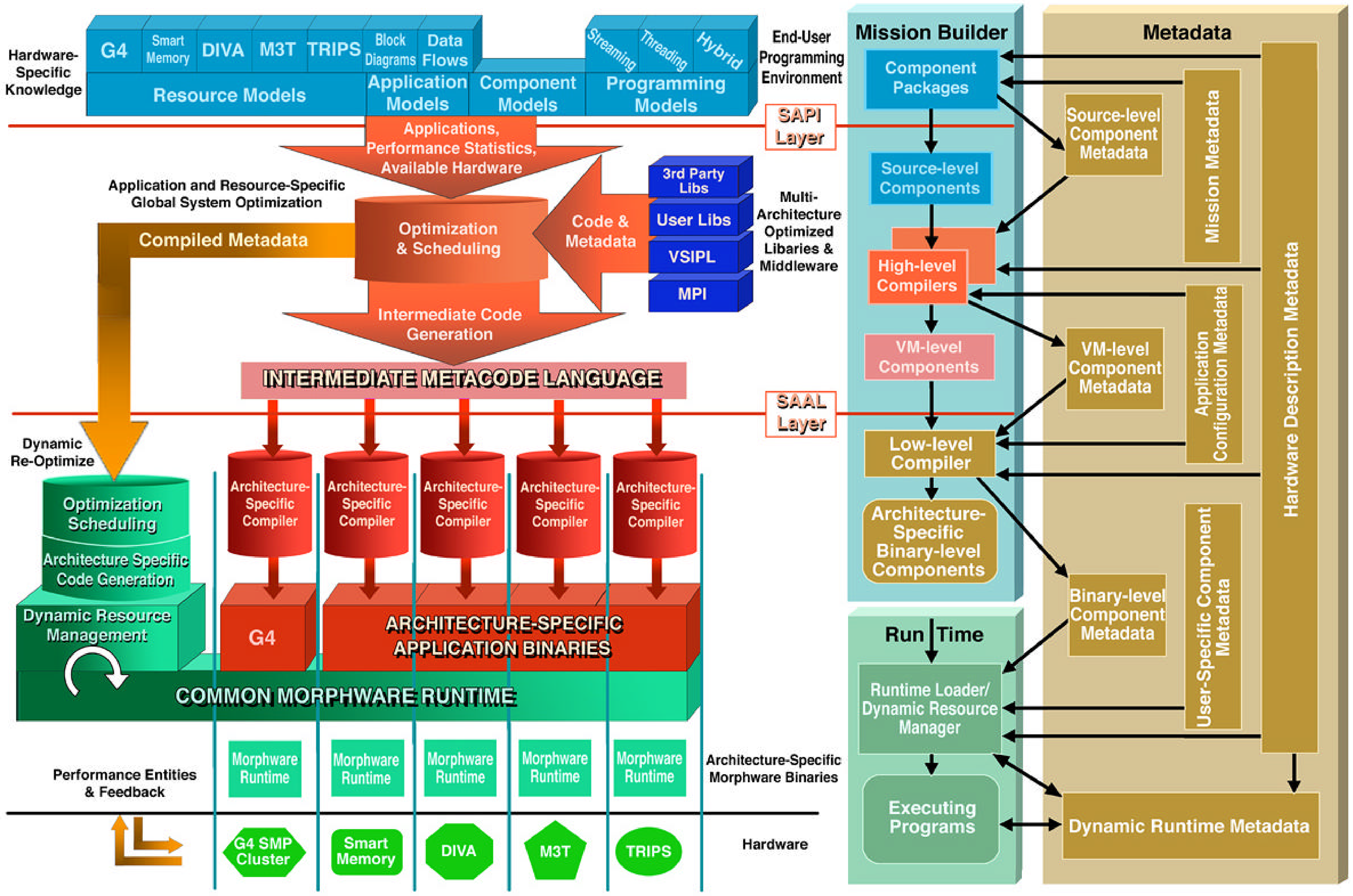
# Goals of the Project

- Study and prototype a modeling language for streaming and threaded resources
- Study and prototype techniques for design space exploration in order to enable PCA scheduling
- Study and prototype techniques for system synthesis and generation
- Contribute findings to the MSI forum

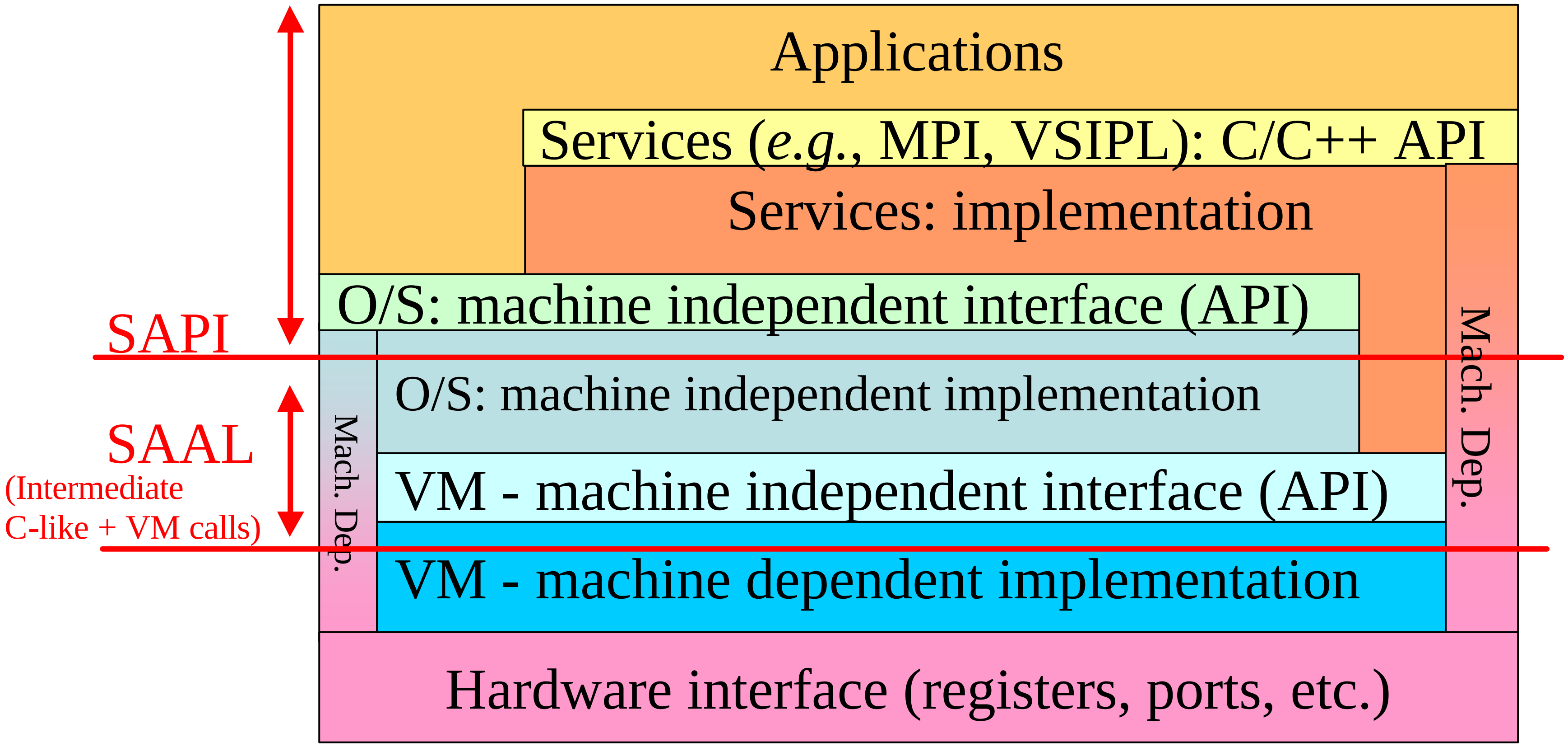
## Organization

- Goals
- **Software Architecture**
- Build Chain
- Metadata
- Dynamic Resource Management
- Software Components
- Prototype Results
- Future Work
- Conclusion

# Integrated PCA Architecture↔Single Metadata System Mapping


Hardware-Specific Knowledge
G4
Smart Memory
DIVA
M3T
TRIPS
Block Diagrams
Data Flows
Resource Models
Application Models
Component Models
Streaming
Threading
Hybrid
Programming Models
End-User Programming Environment
SAPI Layer
Applications, Performance Statistics, Available Hardware
Application and Resource-Specific Global System Optimization
Compiled Metadata
Optimization & Scheduling
Code & Metadata
3rd Party Libs
User Libs
VSIPL
MPI
Multi-Architecture Optimized Libaries & Middleware
Intermediate Code Generation
INTERMEDIATE METACODE LANGUAGE
SAAL Layer
Dynamic Re-Optimize
Architecture-Specific Compiler
Architecture-Specific Compiler
Architecture-Specific Compiler
Architecture-Specific Compiler
Architecture-Specific Compiler
Optimization Scheduling
Architecture Specific Code Generation
Dynamic Resource Management
G4
ARCHITECTURE-SPECIFIC APPLICATION BINARIES
COMMON MORPHWARE RUNTIME
Morphware Runtime
Morphware Runtime
Morphware Runtime
Morphware Runtime
Morphware Runtime
Architecture-Specific Morphware Binaries
Performance Entities & Feedback
G4 SMP Cluster
Smart Memory
DIVA
M3T
TRIPS
Hardware
Mission Builder
Component Packages
Source-level Components
High-level Compilers
VM-level Components
Low-level Compiler
Architecture-Specific Binary-level Components
Run Time
Runtime Loader/ Dynamic Resource Manager
Executing Programs
Metadata
Source-level Component Metadata
Mission Metadata
Hardware Description Metadata
VM-level Component Metadata
Application Configuration Metadata
Binary-level Component Metadata
User-Specific Component Metadata
Dynamic Runtime Metadata

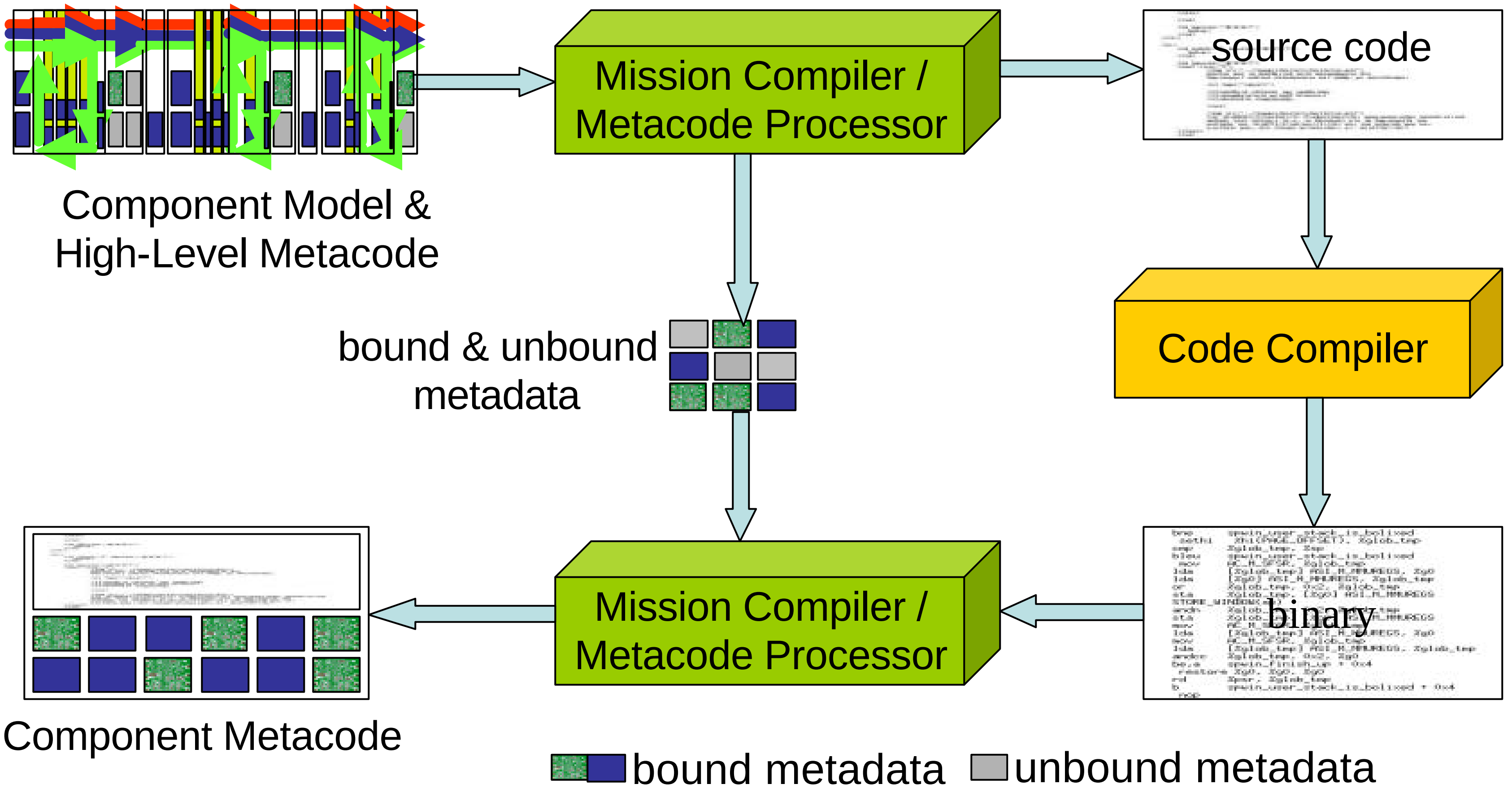
# SAPI, SAAL, VM (C/C++ Application)


Applications
Services (*e.g.*, MPI, VSIPL): C/C++ API
Services: implementation
O/S: machine independent interface (API)
SAPI
O/S: machine independent implementation
SAAL
(Intermediate
C-like + VM calls)
Mach. Dep.
VM - machine independent interface (API)
VM - machine dependent implementation
Mach. Dep.
Hardware interface (registers, ports, etc.)

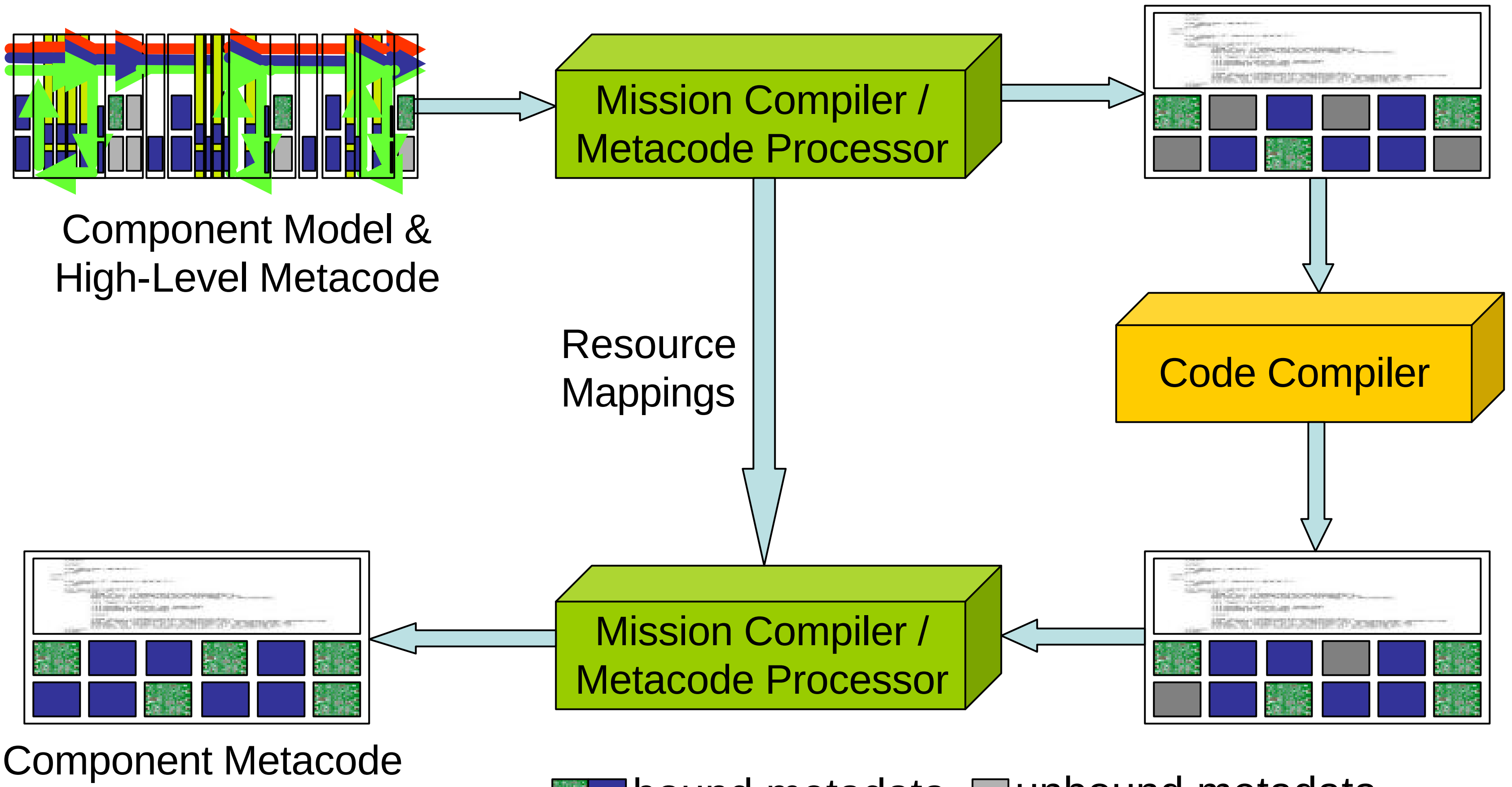
# Organization

- Goals
- Software Architecture
- **Build Chain**
- Metadata
- Dynamic Resource Management
- Software Components
- Prototype Results
- Future Work
- Conclusion

# Current Technology Build Chain

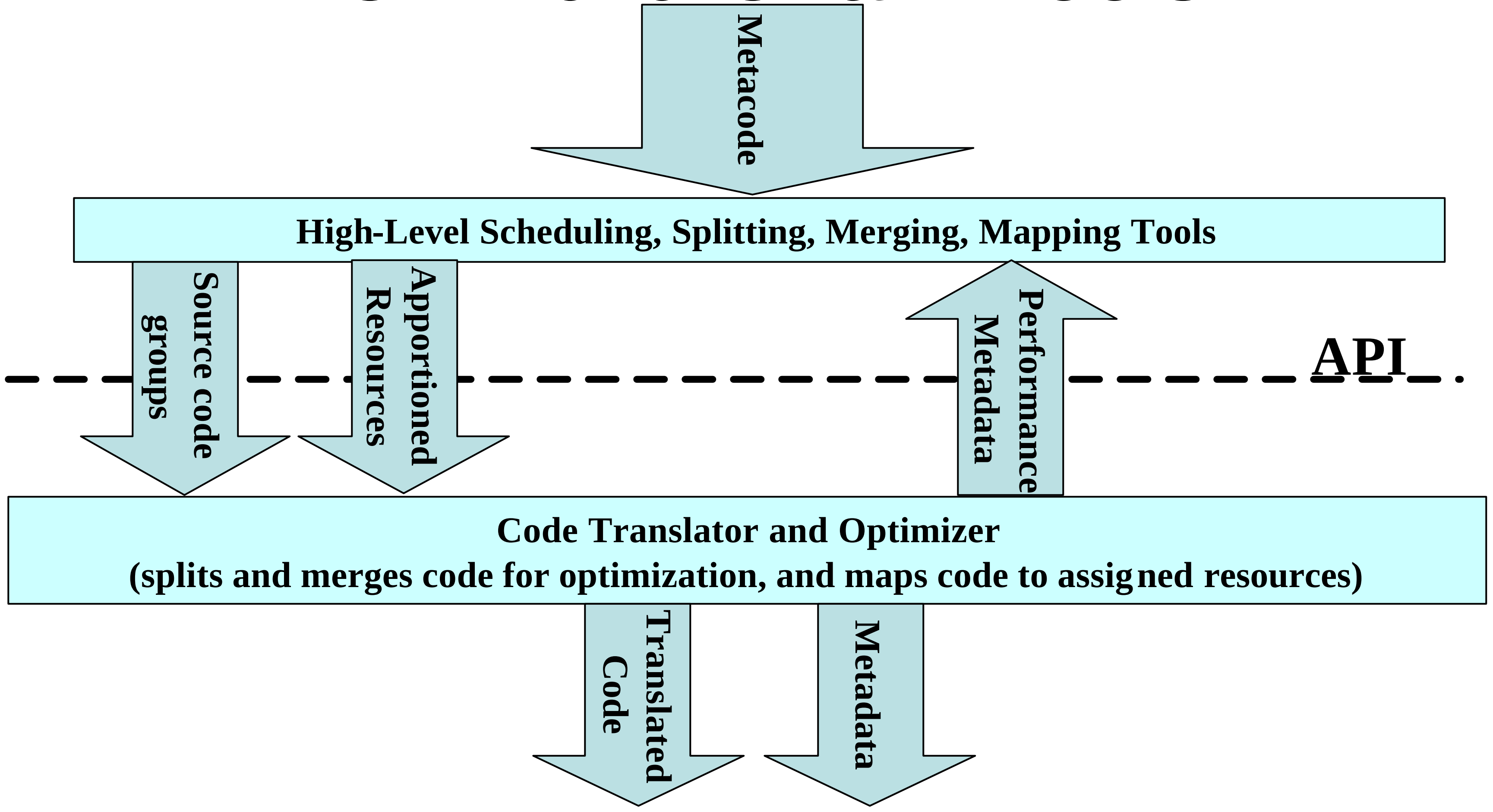

Component Model & High-Level Metacode
Mission Compiler / Metacode Processor
source code
bound & unbound metadata
Code Compiler
binary
Mission Compiler / Metacode Processor
Component Metacode
bound metadata
unbound metadata

# 1st Generation PCA Build Chain



🟩🟦 bound metadata ⬜ unbound metadata

# Interaction of Compilers and PCA Build Chain Tools


Metacode
High-Level Scheduling, Splitting, Merging, Mapping Tools
Source code groups
Apportioned Resources
Performance Metadata
API
Code Translator and Optimizer
(splits and merges code for optimization, and maps code to assigned resources)
Translated Code
Metadata

# Organization

- Goals
- Software Architecture
- Build Chain
- **Metadata**
- Dynamic Resource Management
- Software Components
- Prototype Results
- Future Work
- Conclusion

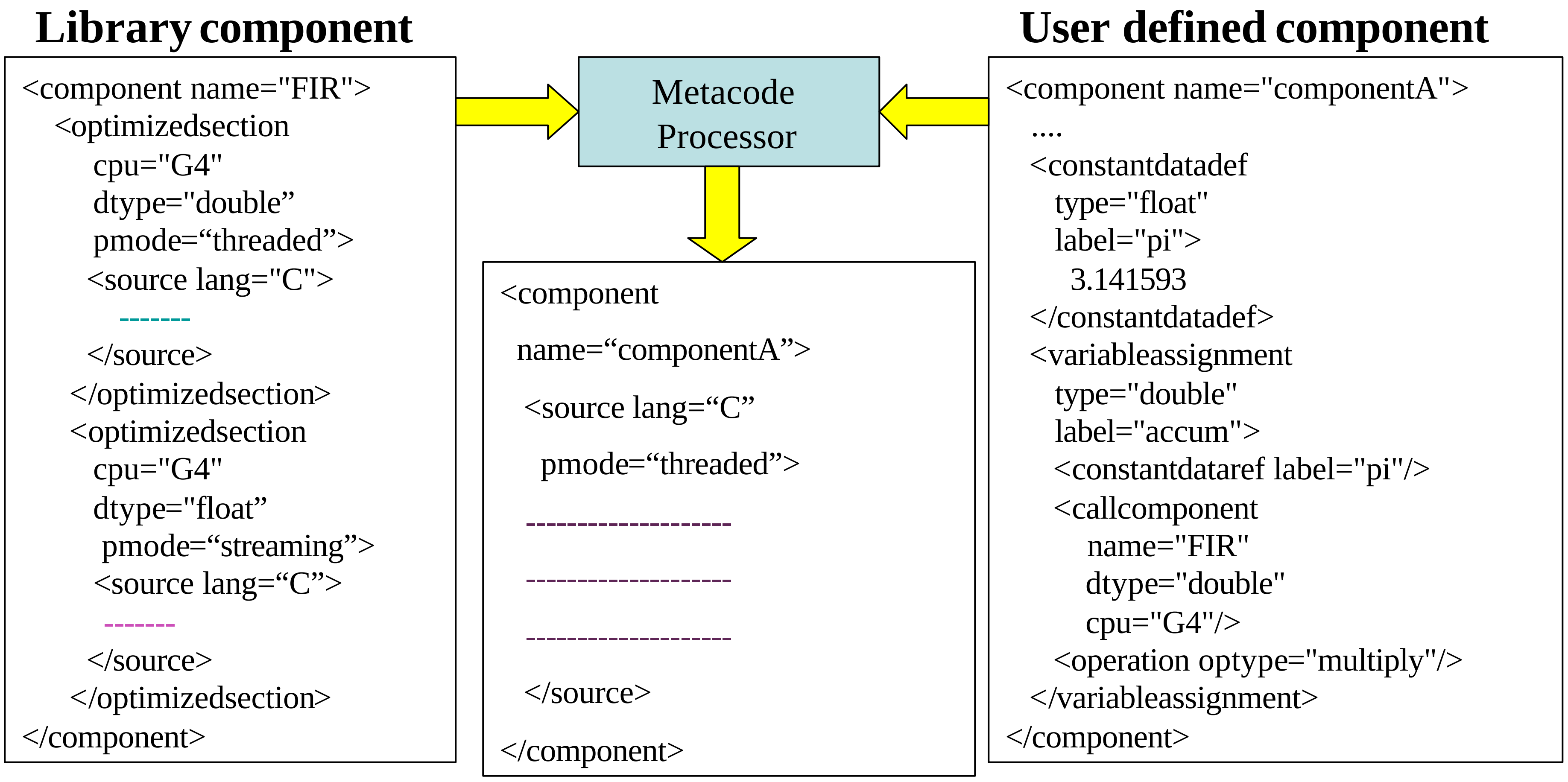
# Metadata

- Describes functionality
  - Machine readable
  - Source code & language details
  - Interface and template parameters
- Known resource requirements
- Known performance capability
- Resource and Performance descriptors:
  - Static table
  - Database query
  - Parametric functions
  - Source code in high-level scripting language

# Determining Metadata

- Derive requirements and capabilities from analysis of source/machine code and hardware capability (Compilers can provide this service)
- Execute code on hardware to verify execution model
- Refine model for every hardware option
  - Analytical modeling will be complex
  - Simplifying assumptions may make model inaccurate
- Construct database of known values
  - Could be large
  - Will be accurate for known hardware configurations
  - Interpolate/extrapolate
- Construct a predictive model
  - Math functions
  - Perl scripts

# XML Metacode



**Library component**

```
<component name="FIR">
   <optimizedsection
       cpu="G4"
       dtype="double”
       pmode=“threaded”>
      <source lang="C">
         --------
      </source>
    </optimizedsection>
    <optimizedsection
       cpu="G4"
       dtype="float”
        pmode=“streaming”>
       <source lang=“C”>
        --------
      </source>
    </optimizedsection>
</component>
```

**User defined component**

```
<component name="componentA">
   ....
   <constantdatadef
      type="float"
      label="pi">
        3.141593
   </constantdatadef>
   <variableassignment
      type="double"
      label="accum">
      <constantdataref label="pi"/>
      <callcomponent
          name="FIR"
          dtype="double"
          cpu="G4"/>
      <operation optype="multiply"/>
   </variableassignment>
</component>
```

```
<component
  name=“componentA”>
   <source lang=“C”
     pmode=“threaded”>
   --------------------
   --------------------
   --------------------
   </source>
</component>
```

# Organization

- Goals
- Software Architecture
- Build Chain
- Metadata
- **Dynamic Resource Management**
- Software Components
- Prototype Results
- Future Work
- Conclusion

# Dynamic Resource Management

- Needed to account for unknown events
  - Faults
  - All possible scenarios may be impossible to predict
  - May be prohibitive to generate static resource & component mappings for all scenarios
- Reduce search space by pre-selecting candidate components at build time

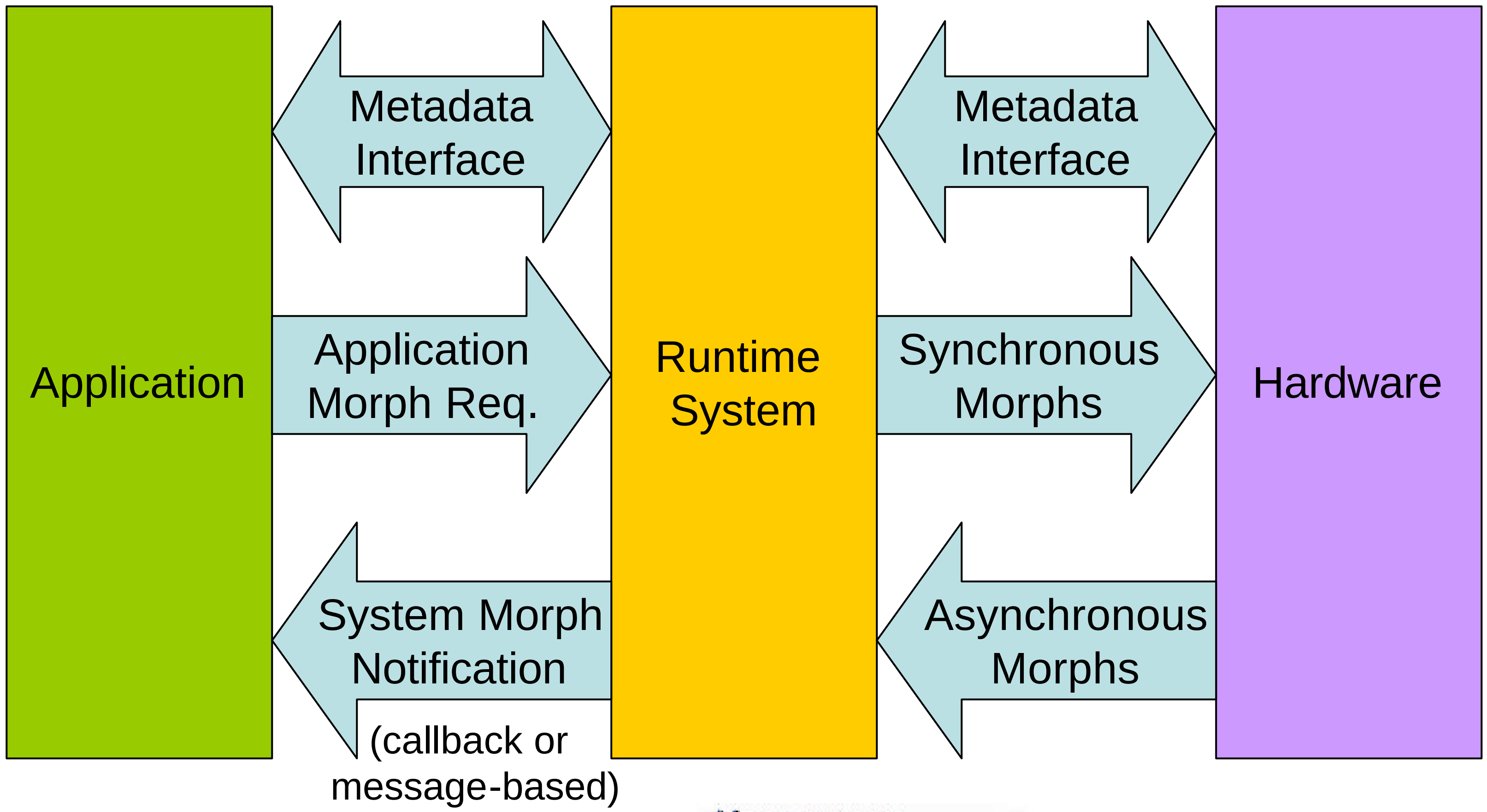
# Application Initiated Morphs

- Application requests resource reconfiguration within allocated resources
  - May be compiler generated
  - May be explicitly coded
- Application explicitly requests new mode (components) within allocated resources
- Application explicitly requests additional resources or releases resources
- Application explicitly requests new mode (components) requiring different resources

# System Initiated Morphs

- System modifies allocated resources transparently (without affecting applications' components)
- System moves application components to execute on a different set of equivalent resources
- System modifies resources allocated to components of an application
- System modifies components and resources allocated to an application

# Application/System Morph Notification and Metadata Interface



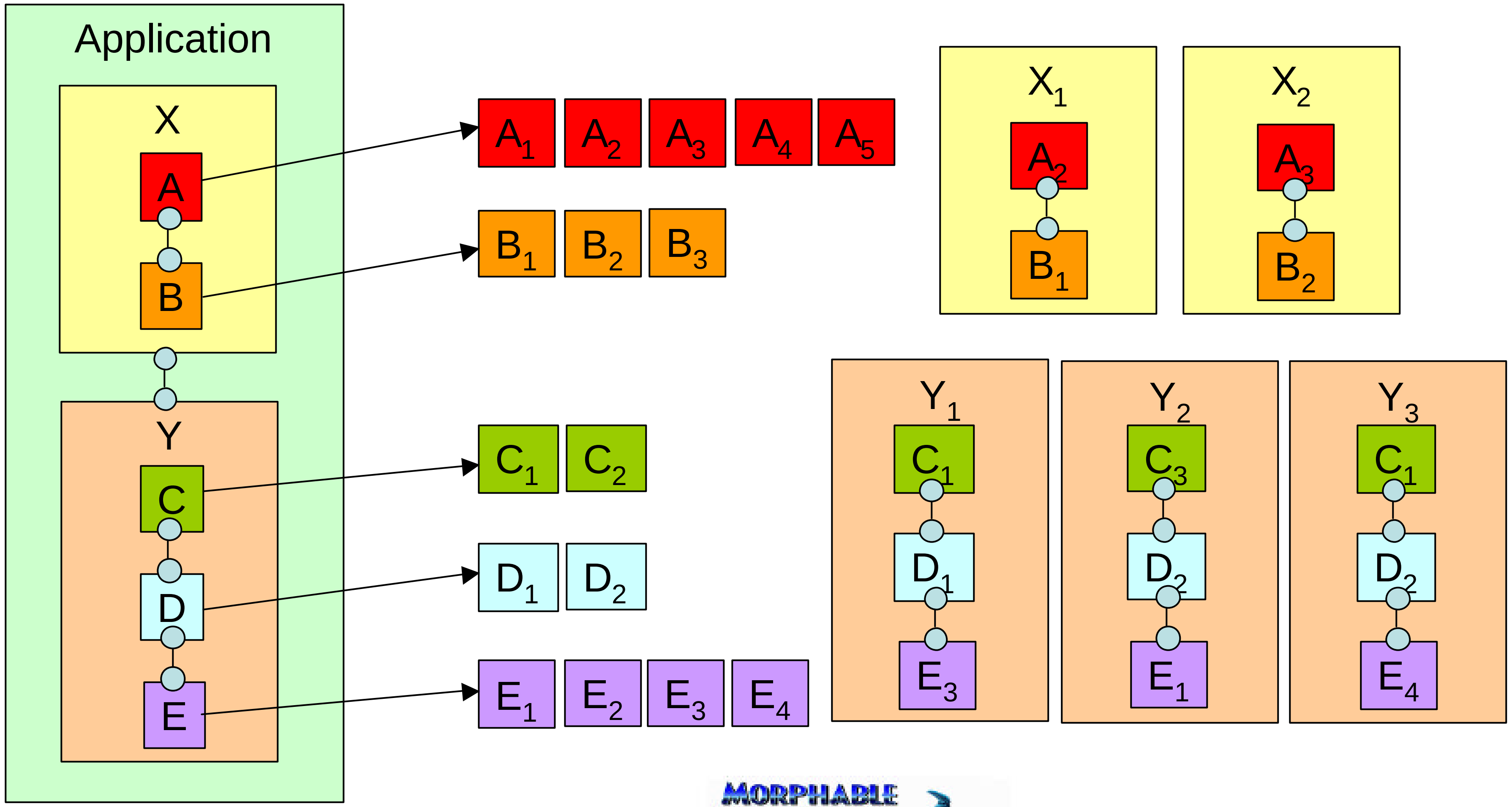
# Organization

- Goals
- Software Architecture
- Build Chain
- Metadata
- Dynamic Resource Management
- **Software Components**
- Prototype Results
- Future Work
- Conclusion

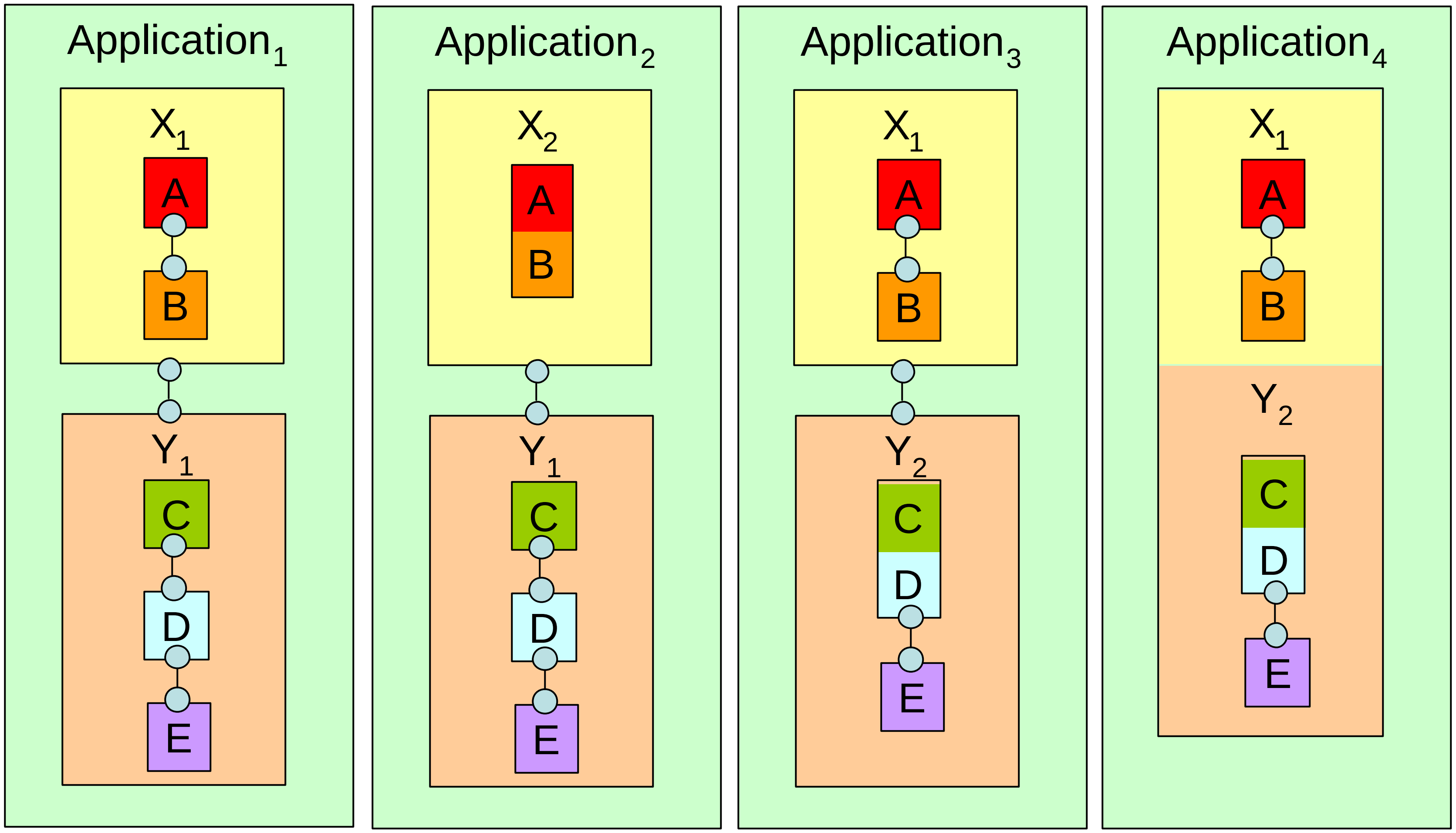
## Component Concepts

- PCA components are hierarchical
- Design and implementation alternatives are explicitly represented in the hierarchy at any level
- A compound component may encapsulate concurrency between lower-level components
- PCA Application is a component composed from a collection of components, component clustering specifications, abstract VM specifications, and component-VM element mappings
- PCA Mission is composed of PCA Applications, VM instantiation, component to physical VM element mappings, mode-change rules, and mission constraints that govern application behaviors

# Component Options



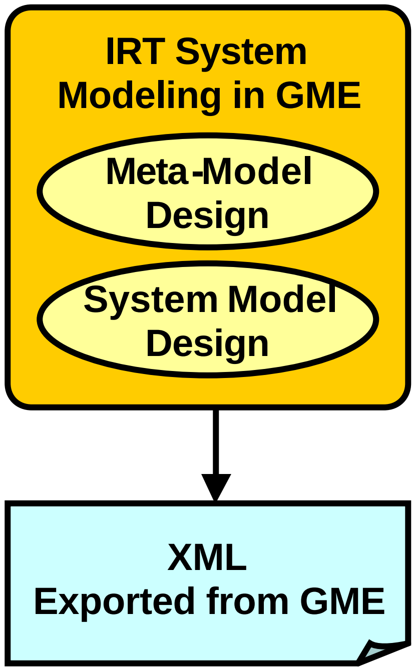
# Component Boundaries



# Organization

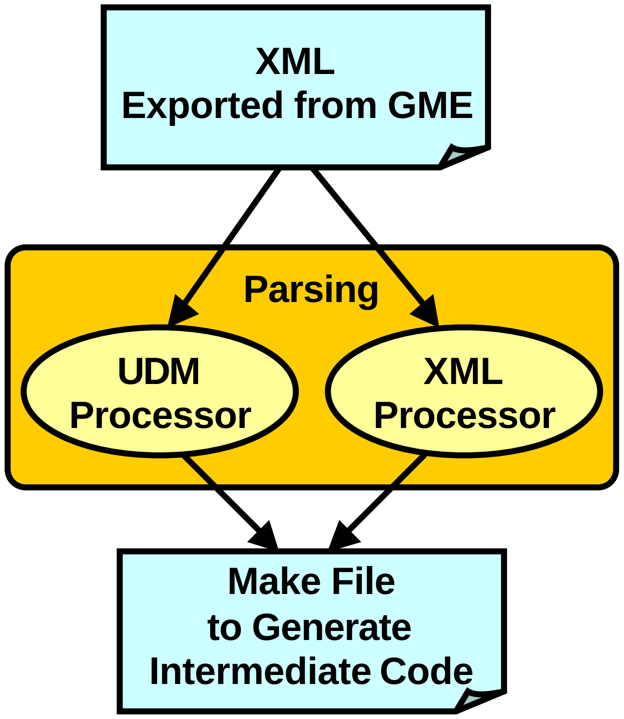
## Step 1: Application Modeling

- System design using GME2000
  - Graphical modeling tool
  - Graphically design component hierarchy
- Define design alternatives
- Specify metadata for each component
- Export model using XML format



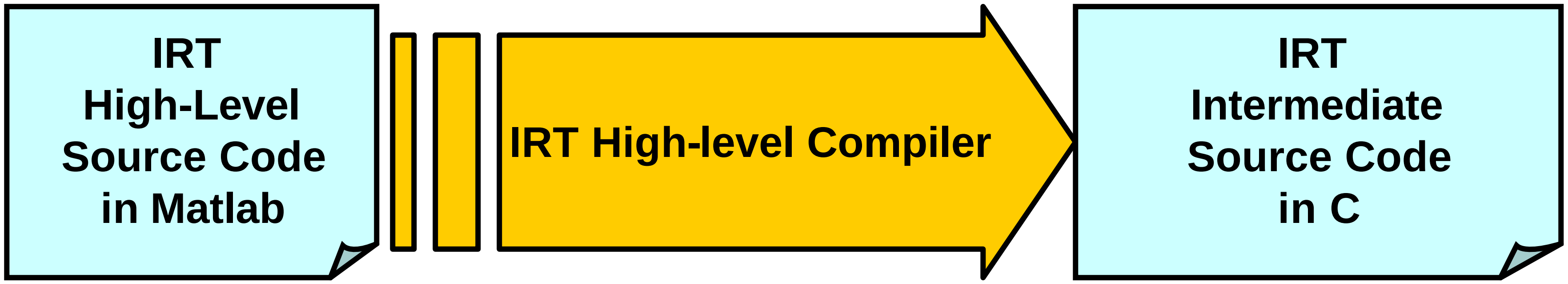
# Step 2: Model Processing

- Parse the XML representation of the model
- Traverse model component hierarchy and retrieve metadata specification of each component
- Dynamically generate makefiles for intermediate code generation and compilation



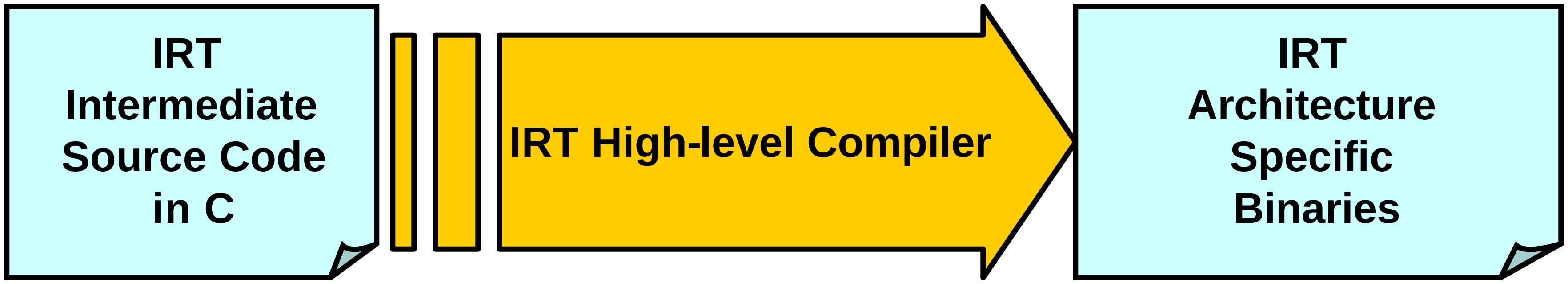
# Step 3: Intermediate Code

- “High-level Compiler”
- Generate the intermediate code for IRT system
- Matlab is end-user programming environment
- C is the intermediate environment
- Translate IRT source from Matlab to C



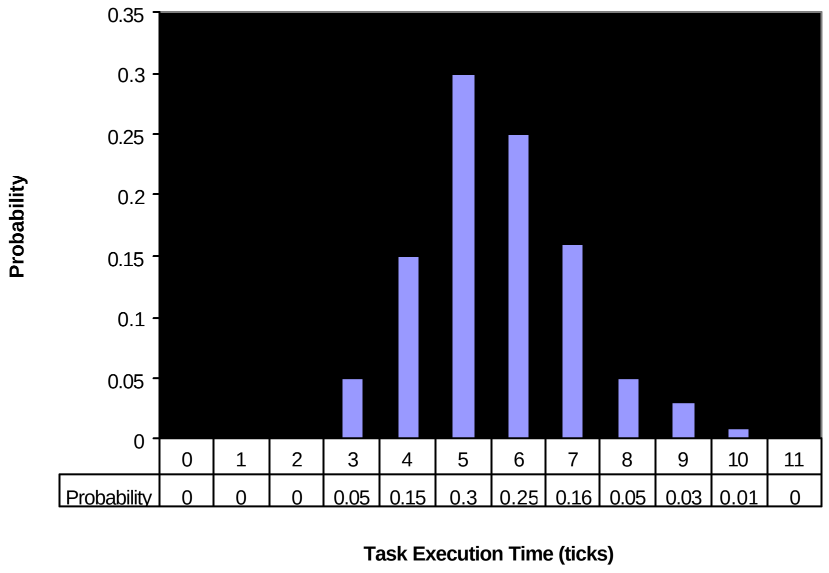
# Step 4: Binaries

- “Low-level Compiler”
- Compile the intermediate code (C code)
- Generate architecture-specific executable binaries
- Metadata is expected to be interpreted and generated for resource management at this level


IRT Intermediate Source Code in C
IRT High-level Compiler
IRT Architecture Specific Binaries

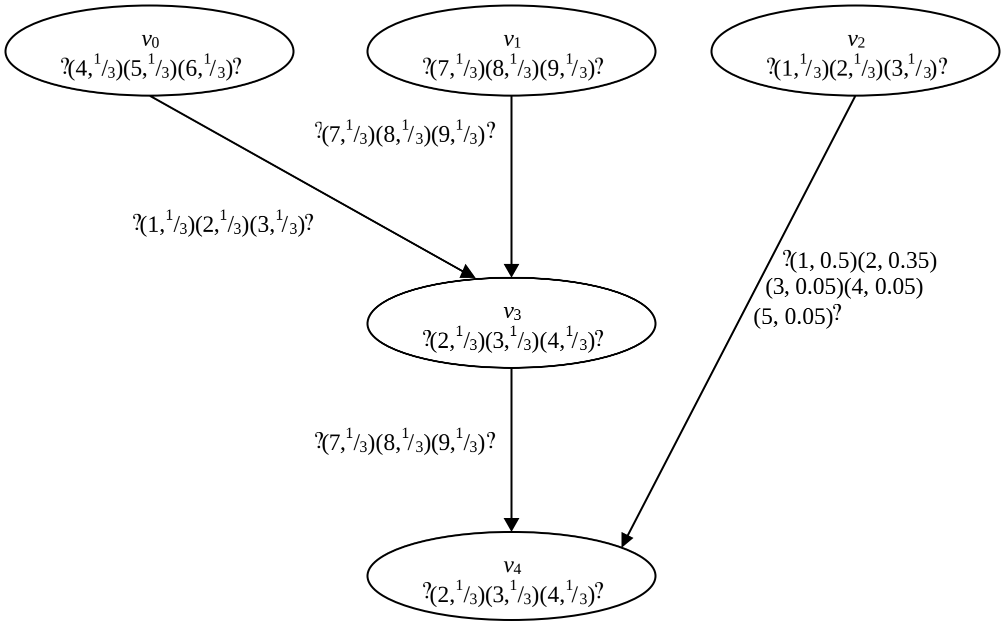
# Scheduling Problem

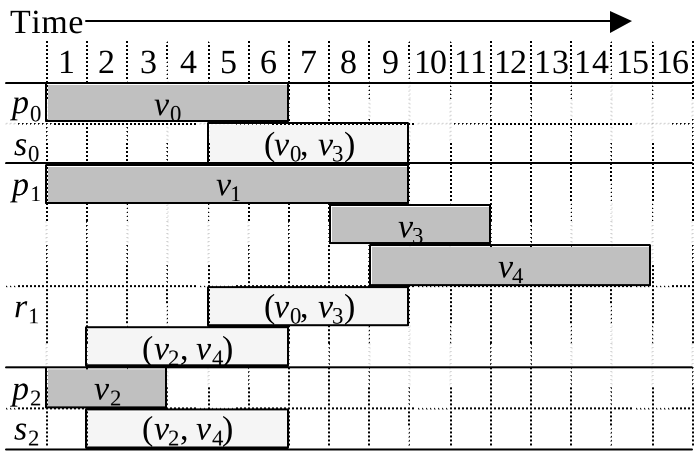
- Non-preemptive schedules for parallel soft real-time applications represented as DAGs
- Metadata specifies execution time probability distributions of tasks (computation and communication)
- Metadata specifies task precedence



| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Probability | 0 | 0 | 0 | 0.05 | 0.15 | 0.3 | 0.25 | 0.16 | 0.05 | 0.03 | 0.01 | 0 |

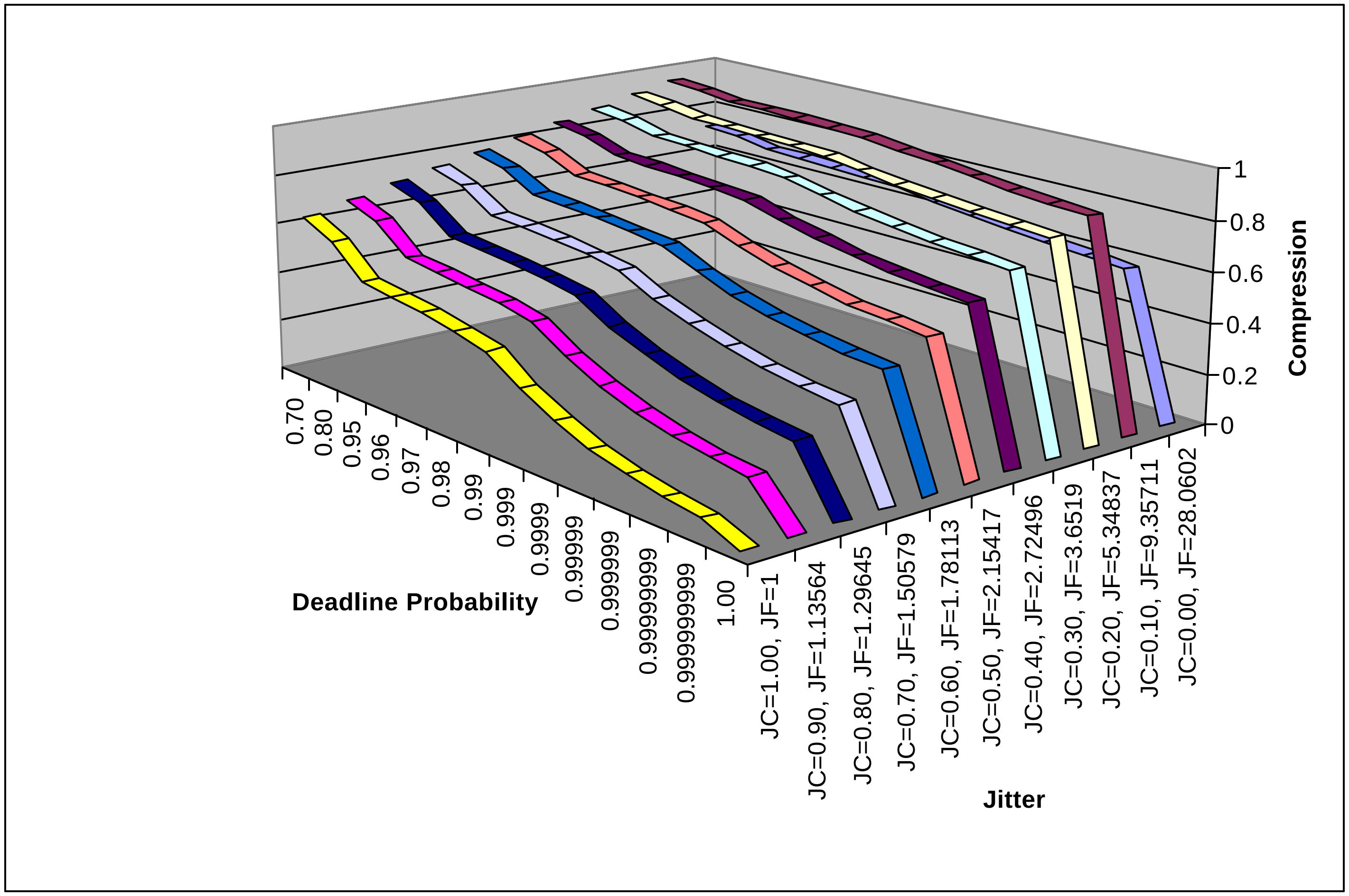
# Schedules





$p_n$: processor $n$
$s_n$: outgoing communication link at processor $n$
$r_n$: incoming communication link at processor $n$

# Scheduling Options



# Organization

- Goals
- Software Architecture
- Build Chain
- Metadata
- Dynamic Resource Management
- Software Components
- Prototype Results
- **Future Work**
- Conclusion

## Future Work - I

- Extend build chain demonstration for a parallel application using MPI on a cluster of dual Xeons with Linux
  - GME2000 metamodel
  - Multiple component implementations with parametric/predictive metadata
  - Metacode parsing and processing
  - Code generation, compilation, and linking
  - Application initiated software morphs
  - Fault handling

# Future Work -II

- Continue to contribute to the MSI forum
  - Metamodeling specification
  - Metadata specification
  - APIs
    - MPI
    - VSIPL
  - Compiler and build chain tool interaction
  - VM specifications

## Organization

- Goals
- Software Architecture
- Build Chain
- Metadata
- Dynamic Resource Management
- Software Components
- Prototype Results
- Future Work
- **Conclusion**

Abstract
Presentation
Back to Agenda
Next Abstract

# Conclusion

- Conceptual architecture for PCA software
- Developed prototype build chain tools
  - Matlab
  - C++, MPI, VSIPL
- Working with the MSI forum to specify
  - Software interfaces
  - Metadata elements and organization
  - Component organization
  - Tool interactions